ROOT+GEANT3 = TGeant

Or why beat a dead horse

Outline

- **Introduction into the problem**
- **motivations**
- **Describing detector geometries**
- **status**
- **milestones**

Problem to solve:

- **How to**
  - Describe detector geometry in the reconstruction code
- **Such that**
  - The same geometry initialization code would be used in the MC code
  - The reconstruction code would not depend on the MC engine (GEANTx or any other) used by the detector simulation code

- **Need to develop appropriate language for describing the geometry**
- **A non-trivial simulation driver to test**

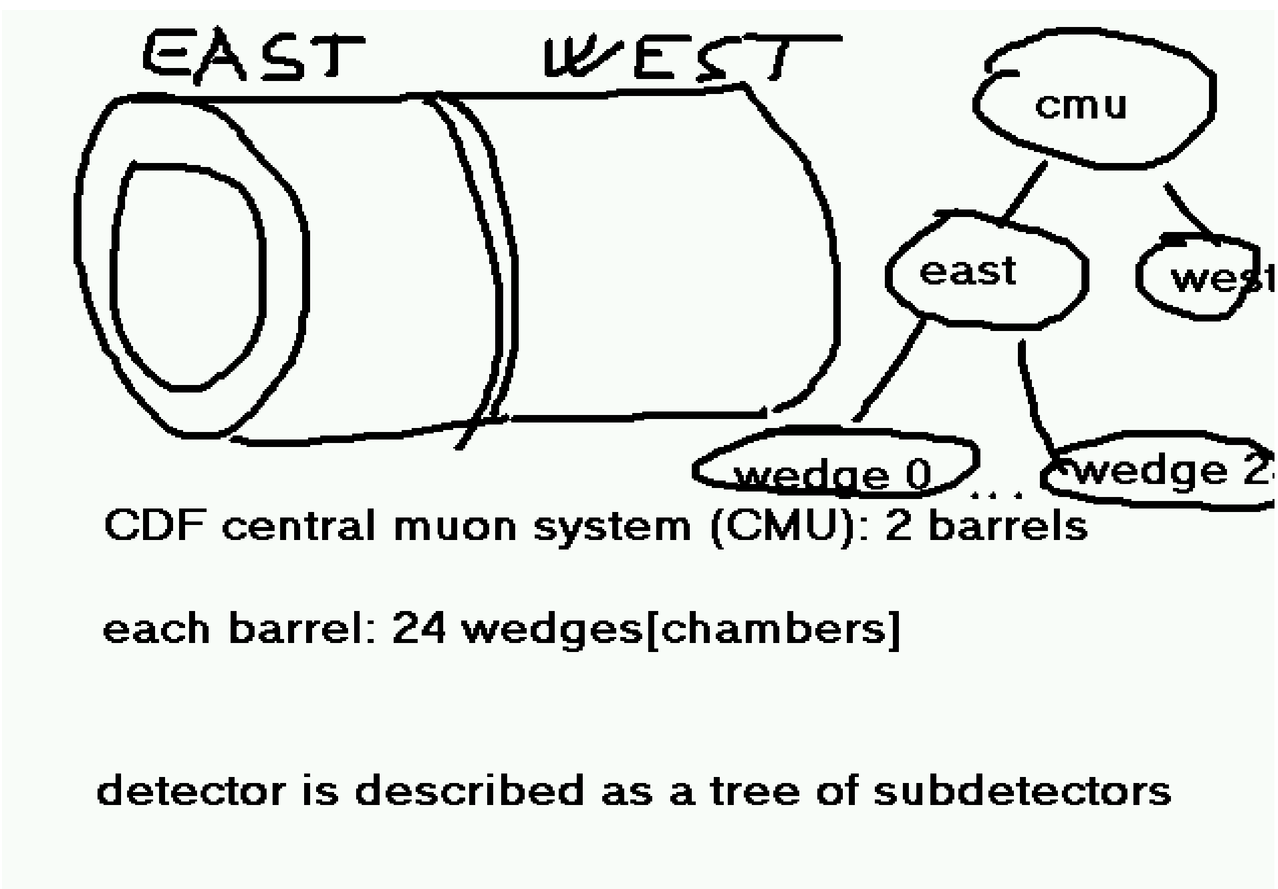
Motivations

(2 cent contribution to the discussion about software design)

- **design by committee:**
  - **decide how to do it, then try to implement and see whether it is possible at all**
- **design by the experts**
  - **design the software system, implement it, give it to users and see what happens (hope for the best)**
- **design by the [unhappy] user**
  - **Do it for yourself and make sure it works for you**
  - **see if what you did can be used by the others**
  - **if yes, share your code with the others**

Geometry declaration: vocabulary

- **when writing C++ code it is especially important to use proper English**
- **(T)DetectorElement:** a simplest part of the detector of interest for the geometry description, I.e. **silicon ladder** or **scintillation counter**
  - Knows about its dimensions, position, mother volume (in the geometry hierarchy)
- **(T)Subdetector :** same as **DetectorElement,** but may have internal structure – daughters - (muon subsystem)



Geometry declaration: procedure

-**Geometry Manager**: provides a set of declaration functions:

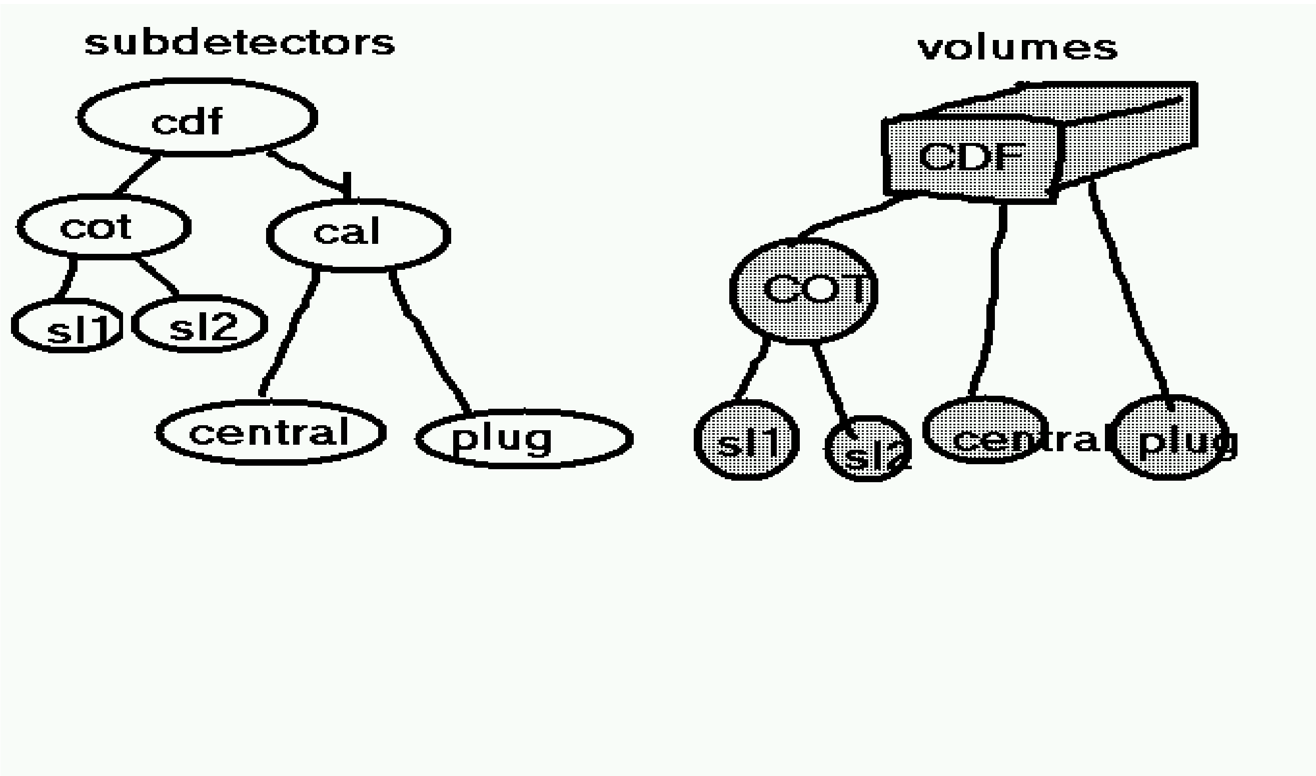
virtual int TGeometryManager::DeclareMaterial(TMaterial*)
virtual int TGeometryManager::DeclareRotation(TRotation*)
virtual int TGeometryManager::CreateShape(…)
virtual int TGeometryManager::CreateVolume(TVolume* v)

-**Key part: Subdetectors declare their geometry to the geometry manager**

TDetectorElement::DeclareGeometry(TGeometryManager*)

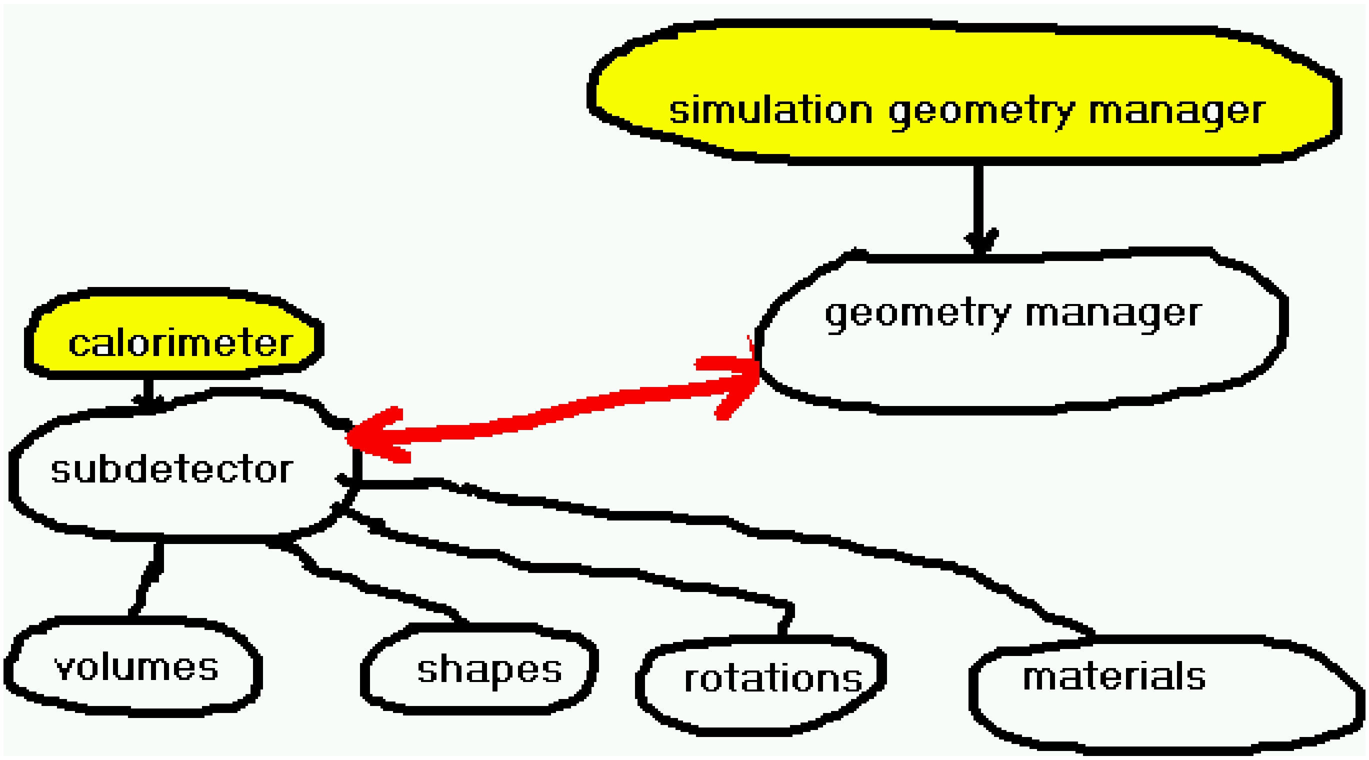
Subdetectors are also responsible for explicit calling geometry declaration routines of their daughters

How it works



•**Subdetector tree: explicit (description of the CDF detector, for example)**
•**Volume tree: generic description of the detector geometry (volumes, materials, tracking media etc)**

Geometry declaration:
including the simulation



-all the interaction between the [experiment-specific] geometry initialization code and the geometry management system goes through the interaction between 2 classes: GeometryManager and the DetectorElement

•simulation geometry manager inherits from the base geometry manager class and overloads its virtual declaration methods

•Pass the simulation geometry manager to the top node of the subdetector tree instead of the base class:

Top->DeclareGeometry(TGeometryManager*)

Geometry declaration: what is implemented

-generic shape, generic volume (extend TShape/TVolume to add what is necessary for the MC needs)

-Several shapes (what was needed to describe pieces of the CDF detector) implemented: box, tube, trapezoidal

-Supported:

  -Volume sub-tree copies

  -Volume divisions

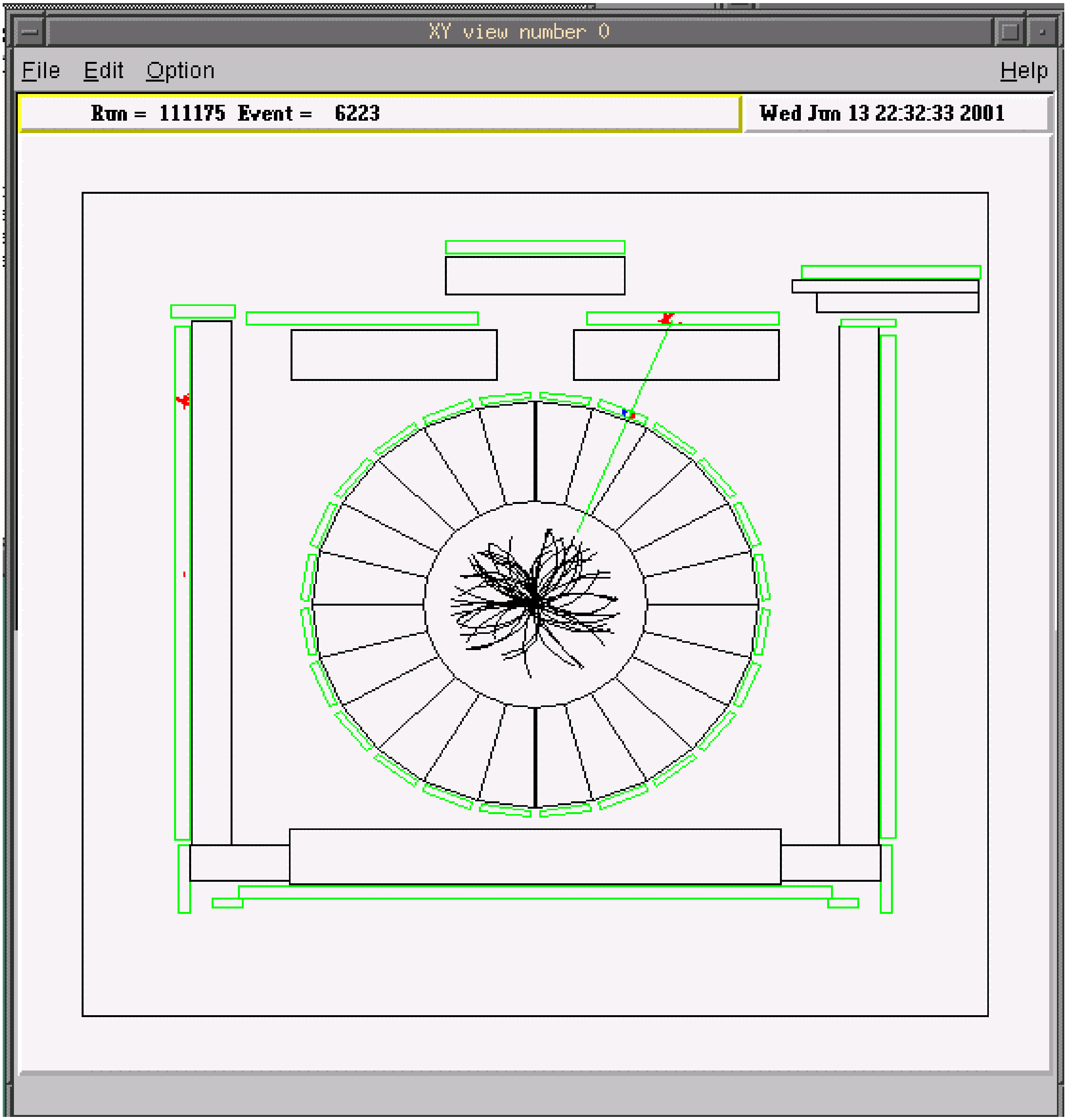
  -Bolean operations (define the new shape)

  -Support for overlapping volumes: to come

-Generic class for the geometry manager

-Generic class for visualization manager

- First rule of the “design by the user”: **make sure that it works for you**



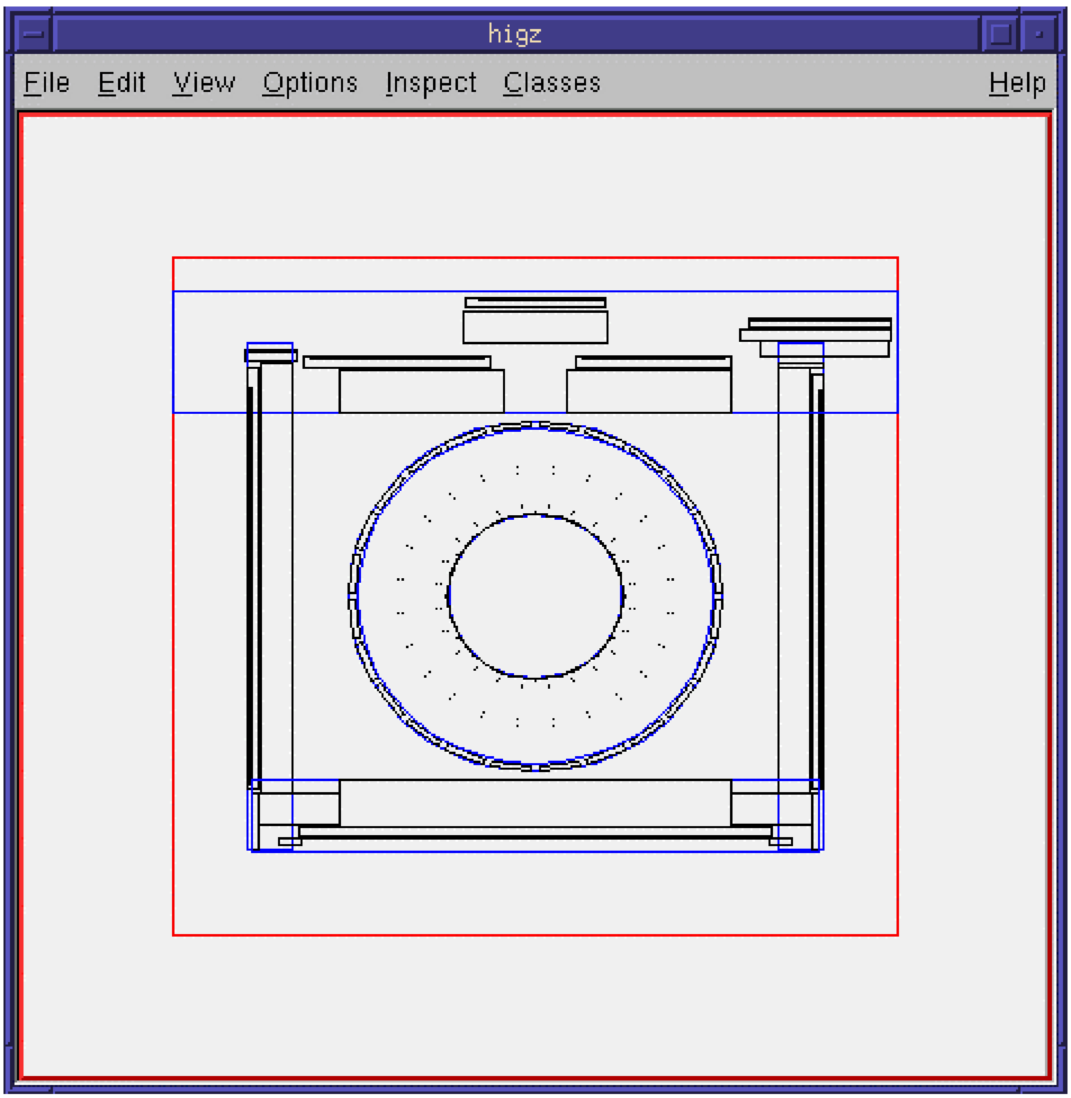
While you eat, your appetite grows up

- **Geometry initialization scheme works well for the reconstruction code**
- **To make a real test with the simulation need a real simulation driver**
- GEANT3 was a 1st widely used in HEP simulation/reconstruction/analysis environment – has all the hooks
- It is by far the best understood general purpose MC code
- Including all its limitations and problems

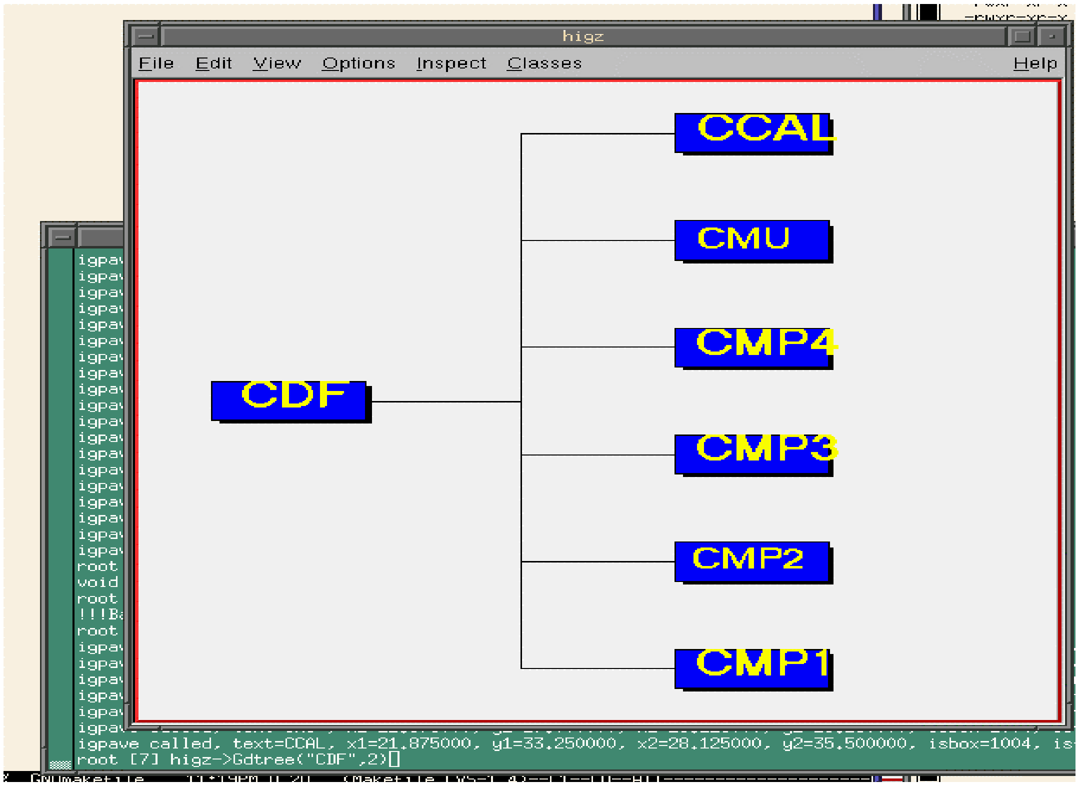
- **Why not to start from Geant3?**

Geometry manager for GEANT3

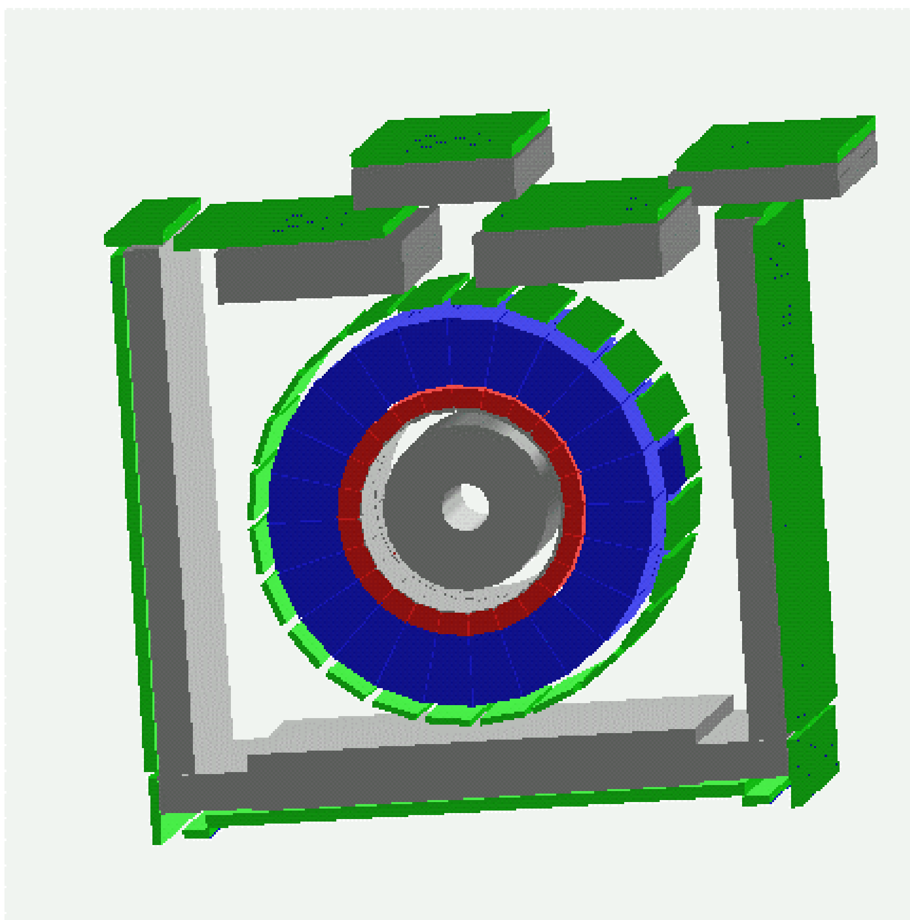
- Implement **TGeant3GeometryManager** (inherits from **TGeometryManager)**
- **Play with THigz class** (a C++ wrapper around HIGZ by Rene), make it inheriting from generic visualization manager (GEANT3-specific implementation)
- Pass TGeant3GeometryManager to the code initializing CDF geometry:



So the appetite grows up …



And grows up …



How dead is the horse?

GEANT3 issues

- **Monolithic:**
  - Cross-dependencies between the sub-packages
  - explicit knowledge of the format of ZEBRA structures is assumed in many places
- **ZEBRA: 32 bit-long representation of floating point numbers**
- **FORTRAN inheritance: needs all of the CERNLIB**
- **Physics: far from the best (Atlas note PHYS-no-086)**
  - Hadronic interactions: last standalone version of FLUKA does much better job than GFLUKA
  - EM processes: thin gas layers

What would a user really like to have implemented?

- floating point numbers storage in double precision (get rid of ZEBRA)
- Split GEANT3 code into pieces by functionality (separate physics process management from geometry)
- Migrate to C++
- Implement ROOT/CINT-based interactive interface (analysis environment)
- Work on improving the physics – w/o (1) and (2) it is not very likely to happen

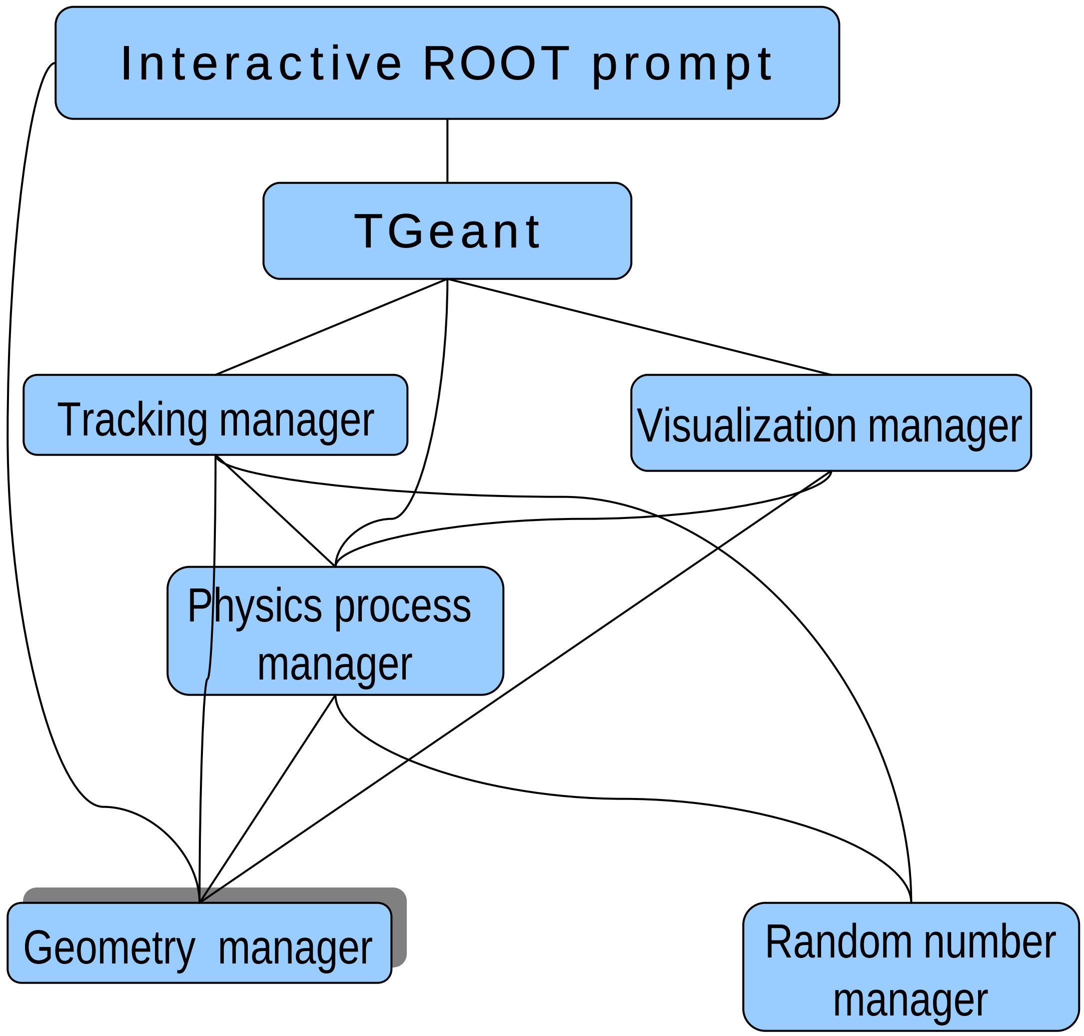
- Many experiments (ALICE, CDF, STAR, Hera-B, Phobos, D0 and ,I believe, the others) put C++ interface on top of GEANT3
- This allowed to continue using GEANT3
- Didn't solve the major problems

Split GEANT3 code into the modules
Done Jan'2001

```
// load shared libraries needed to run TGEANT
void start() {

  gSystem->Load("AliRoot/lib/tgt_Linux/libminicern.so");
  gSystem->Load("lib/$BFARCH/libgeant321_utils.so");
  gSystem->Load("lib/$BFARCH/libgeant321_ggeom.so");
  gSystem->Load("lib/$BFARCH/libgeant321_gheisha.so");
  gSystem->Load("lib/$BFARCH/libgeant321_gcons.so");
  gSystem->Load("lib/$BFARCH/libgeant321_gphys.so");
  gSystem->Load("lib/$BFARCH/libgeant321_gtrak.so");
  gSystem->Load("lib/$BFARCH/libgeant321_gbase.so");

                    // will work in interactive mode...

  gSystem->Load("lib/$BFARCH/libgeant321_gdraw.so");
//-----------------------------------------------------------------------
```

- Modularity achieved at the level of algorithms
- But not at the level of the data structures

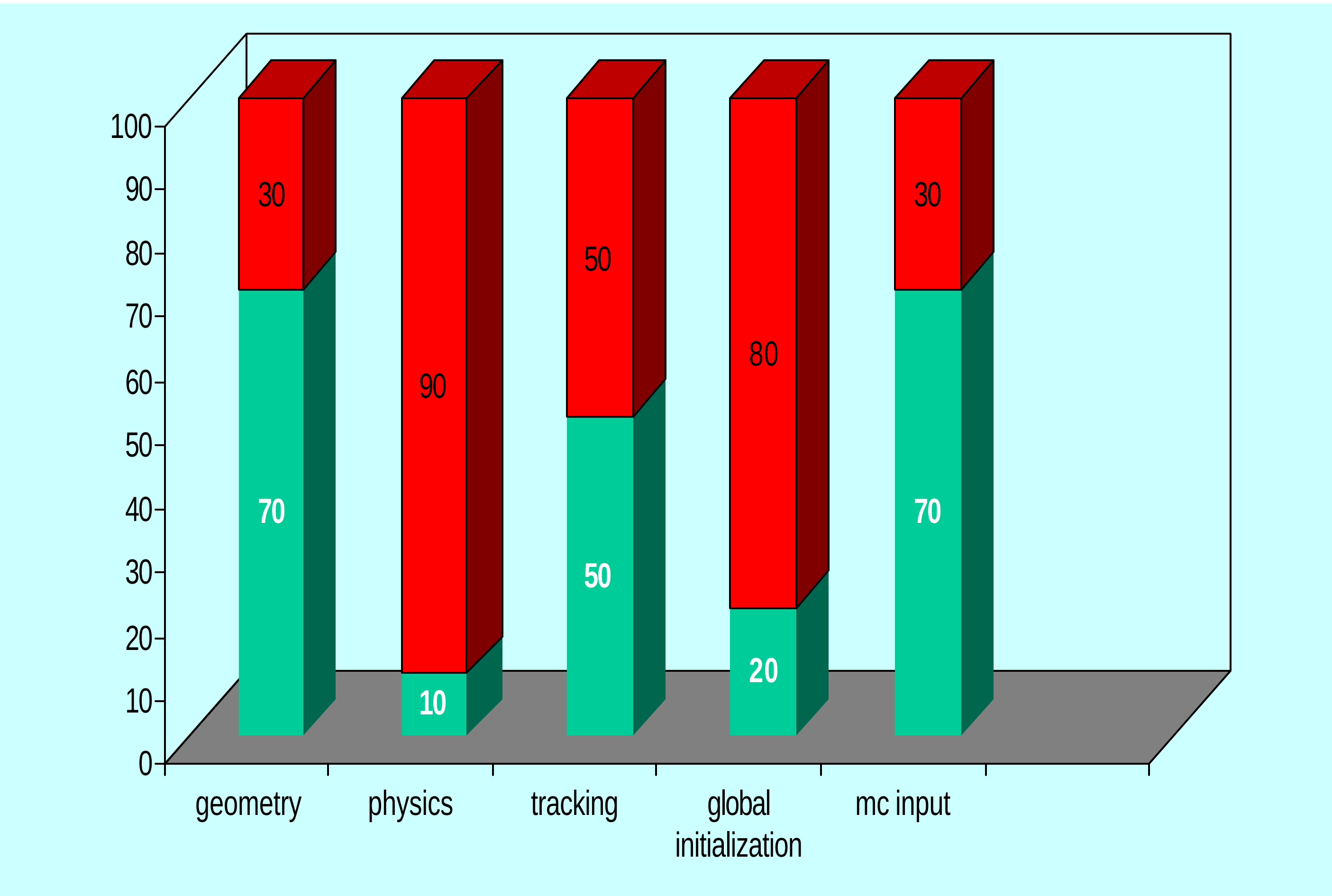
Steps: Implement C++ interfaces between the (still FORTRAN) modules



Implementation of the managers: "semi-abstract" interface, base classes provide minimal default implementation

- Replace ZEBRA structures one by one with the ROOT-based code providing the same functionality
- In theory: can transition from all FORTRAN to all C++ adiabatically
- In practice: can make it up to a certain point
- Pretty soon realize that to make the next step, need to take everything apart first

Where the project stands



Where the project goes:

- First big milestone: fall'01
  - Pieces brought back together
  - functionality of GEANT 3.21 restored
  - Management code migrated to C++
  - No ZEBRA
  - FORTRAN still used (to simulate physics processes, for example)
  - start validation and timing tests

Wish list: documentation

- Logically shouldn't need dictionaries for autodoc generation
- Same class described in several source files: still want to be able to generate documentation
- Way of documenting inline functions